AI TRAVEL GUIDE

PROIECT PROIECTAREA APLICAȚIILOR WEB

Autor: Salnicean Razvan

Programul de studii: CALCULATOARE

Cuprins

1 Introducere

1.1 Context general

În contextul actual al globalizării și al digitalizării accelerate, industria turismului trece printr-o transformare fundamentală. Modul în care călătorii își planifică vacanțele și explorează destinațiile s-a schimbat radical odată cu apariția tehnologiilor web moderne și a dispozitivelor mobile inteligente. Ghidurile turistice tradiționale, tipărite pe hârtie, și hărțile fizice statice au devenit relicve ale trecutului, fiind înlocuite treptat de soluții digitale care oferă interactivitate, actualizări în timp real și un conținut vizual bogat.

Aplicația web **AI Travel Guide**, dezvoltată în cadrul acestui proiect, s-a născut din nevoia de a oferi utilizatorilor o modalitate eficientă, modernă și scalabilă de a descoperi destinații turistice. Într-o lume în care informația este omniprezentă, provocarea nu mai constă în găsirea datelor, ci în filtrarea, agregarea și prezentarea lor într-un mod relevant și ușor de asimilat. Turiștii moderni nu doresc doar o listă de nume de muzee sau parcuri; ei doresc să vizualizeze spațial aceste obiective, să vadă imagini reprezentative instantaneu și să înțeleagă contextul geografic al destinației lor înainte de a ajunge acolo.

Mai mult, integrarea datelor din surse multiple a devenit un standard în industrie. Utilizatorii se așteaptă ca o singură platformă să le ofere atât localizarea pe hartă, cât și descrieri enciclopedice, fără a fi nevoiți să navigheze între o aplicație de hărți (precum Google Maps) și un motor de căutare sau o enciclopedie online (precum Wikipedia). Proiectul de față propune o abordare integrată, combinând puterea sistemelor de informații geografice (GIS) accesibile prin web cu baze de date dinamice și servicii de notificare în timp real. Această sinergie tehnologică transformă simpla planificare a unei călătorii dintr-o sarcină logistică într-o experiență de explorare virtuală captivantă.

De asemenea, aspectul "live" al aplicațiilor moderne este crucial. Într-un mediu urban dinamic, informațiile se schimbă rapid: un obiectiv poate fi închis temporar, un eveniment spontan poate avea loc într-o piață centrală sau condițiile meteo pot afecta vizitarea anumitor obiective. Prin implementarea unor mecanisme de comunicare în timp real (WebSockets), aplicația *AI Travel Guide* își propune să depășească statutul de catalog static și să devină un asistent activ pentru utilizator, capabil să trimită alerte și sugestii contextuale.

1.2 Obiective

Scopul fundamental al acestei aplicații web este de a ușura cât mai mult procesul de documentare și explorare turistică, oferind o platformă centralizată unde informația vizuală (hărți) se îmbină armonios cu informația descriptivă.

Prin dezvoltarea acestei aplicații se urmărește atingerea următoarelor obiective specifice:

- **Facilitarea Explorării Vizuale:** Implementarea unei hărți interactive (folosind tehnologii precum Leaflet și OpenStreetMap) care să permită utilizatorilor să vizualizeze distribuția spațială a atracțiilor turistice. Obiectivul este ca utilizatorul să poată identifica rapid zonele de interes maxim dintr-un oraș, nu doar să citească o listă text.

- **Agregarea Automată a Informațiilor:** Integrarea cu API-uri externe de încredere (precum Wikipedia API și servicii de geocodare Nominatim/Photon) pentru a popula automat aplicația cu date relevante: imagini, descrieri istorice și coordonate precise. Acest obiectiv asigură scalabilitatea aplicației, eliminând necesitatea introducerii manuale a datelor pentru fiecare oraș din lume.

- **Interactivitate în Timp Real:** Dezvoltarea unui sistem de notificări "push" bazat pe protocolul WebSocket. Acest sistem are rolul de a simula prezența unui ghid turistic virtual care poate alerta utilizatorul cu privire la sugestii de moment, avertizări meteo sau promoții locale, fără a necesita reîncărcarea paginii.

- **Experiență de Utilizare Fluidă (UX):** Crearea unei interfețe de tip Single Page Application (SPA) folosind React, care să răspundă instantaneu la comenzile utilizatorului (căutare, zoom, click pe marker). Prioritatea este minimizarea timpilor de așteptare și oferirea unui feedback vizual clar pentru fiecare acțiune.
- **Gestiunea Securizată a Accesului:** Implementarea unui sistem de autentificare robust (Login/Register) folosind Spring Security, pentru a permite diferențierea între vizitatori și utilizatorii înregistrați, punând bazele pentru funcționalități viitoare de personalizare (ex: salvarea favoritelor).

1.3 Lista Moscow

În cadrul procesului complex de dezvoltare a aplicației de ghid turistic inteligent, gestionarea resurselor și a timpului a fost critică. Pentru a asigura livrarea unui produs funcțional și robust, am utilizat metoda de prioritizare **MoSCoW** (Must have, Should have, Could have, Won't have). Această metodă a permis clasificarea cerințelor funcționale și non-funcționale în funcție de impactul lor asupra produsului final.

Analiza cerințelor a rezultat în următoarea structură de priorități:

- **Must-haves (Trebuie să aibă):** Acestea sunt funcționalitățile critice fără de care aplicația nu ar avea sens. Includem aici capacitatea de a căuta un oraș, afișarea hărții interactive, plasarea markerelor pentru atracții și existența unui backend care să servească aceste date.
- **Should-haves (Ar trebui să aibă):** Funcționalități importante care adaugă valoare semnificativă. Aici intră integrarea cu Wikipedia pentru imagini și descrieri detaliate, sistemul de notificări în timp real și autentificarea utilizatorilor.
- **Could-haves (S-ar putea să aibă):** Funcționalități dezirabile, de tip "nice-to-have", care îmbunătățesc experiența dar nu sunt vitale pentru lansare. Exemple: filtrarea atracțiilor după categorie, istoricul notificărilor sau localizarea utilizatorului pe hartă ("You are here").
- **Won't-haves (Nu va avea):** Funcționalități care au fost excluse explicit din această versiune a proiectului pentru a menține focusul, cum ar fi sistemul de plăți, rezervări hoteliere sau integrarea cu rețele sociale complexe.

Must have	**Should have**	**Could have**	**Won't have**
Căutare orașe și afișare pe hartă	Integrare API Wikipedia (imagini și text)	Filtrare avansată a atracțiilor	Sistem de rezervări zboruri/hotel
Backend RESTful și Bază de Date	Notificări în timp real (WebSocket)	Istoric al notificărilor	Procesare plăți online
Afișare markere și popup-uri informative	Autentificare (Login/Register)	Localizare GPS a utilizatorului	Chatbot AI complex

Tabela 1: Lista MoSCoW pentru AI Travel Guide

1.4 Cazuri de utilizare

Aplicația *AI Travel Guide* este destinată în principal pasionaților de călătorii (Turiști), dar include și funcționalități de sistem pentru administrarea fluxului de informații. În cadrul aplicației am identificat diverse cazuri de utilizare care reflectă scenariile principale de interacțiune. Acestea ilustrează modul în care utilizatorii interacționează cu harta, cu sistemul de căutare și cu notificările.

Principalele scenarii identificate sunt:

1. **Căutarea și Explorarea unei Destinații:** Acesta este scenariul central al aplicației. Utilizatorul accesează pagina principală și introduce numele unui oraș (de exemplu, "Paris") în bara de căutare. Sistemul procesează cererea, identifică coordonatele orașului prin servicii de geocodare, încarcă harta zonei respective și populează automat harta cu markere reprezentând principalele atracții turistice stocate în baza de date sau preluate din surse externe.

2. **Vizualizarea Detaliilor Obiectivelor Turistice:** Odată ce atracțiile sunt afișate pe hartă, utilizatorul dorește să afle mai multe informații. Prin simpla apăsare (click) pe un marker personalizat, se deschide o fereastră de tip "popup". Aceasta afișează o imagine reprezentativă a obiectivului (preluată dinamic de pe Wikipedia), numele acestuia și un link direct către articolul enciclopedic complet. Acest caz de utilizare transformă harta dintr-un instrument de orientare într-un instrument de învățare.

3. **Recepționarea Notificărilor în Timp Real:** În timp ce utilizatorul navighează pe site, sistemul poate trimite alerte contextuale. De exemplu, o notificare de tip "Toast" poate apărea în colțul ecranului cu un mesaj precum "Atenție: Ploaie torențială în centrul orașului!". Utilizatorul vede această notificare instantaneu, fără a fi nevoie să dea refresh la pagină, și poate consulta un istoric al acestor alerte prin intermediul meniului de notificări (clopoțel).

4. **Autentificarea în Sistem:** Pentru a accesa funcționalități personalizate sau pentru a avea o experiență securizată, utilizatorul poate alege să își creeze un cont sau să se autentifice. Scenariul implică completarea unui formular de Login/Register, validarea credențialelor de către backend și acordarea accesului pe baza token-ului de sesiune.

5. **Identificarea Locației Curente (Geocodare Inversă):** Utilizatorul poate dori să vadă unde se află pe hartă sau să afle adresa unui punct arbitrar. Prin interacțiunea cu harta (click), aplicația plasează un marker distinct ("You are here!") și, folosind geocodarea inversă, îi spune utilizatorului adresa exactă sau numele cartierului selectat.

2 Arhitectura

Arhitectura aplicației **AI Travel Guide** este concepută pentru a asigura o performanță de înaltă calitate, scalabilitate și o interfață utilizator prietenoasă, esențială pentru o aplicație vizuală bazată pe hărți. Am optat pentru o arhitectură de tip *Client-Server* decuplată (Decoupled Architecture), unde logica de prezentare este complet separată de logica de business și persistența datelor.

Cu un front-end dezvoltat în **React** și un backend bazat pe **Spring Boot**, această arhitectură modernă îmbină tehnologii de ultimă generație pentru a oferi o experiență coerentă și robustă utilizatorilor. Această separare permite dezvoltarea independentă a celor două componente, facilitând întreținerea și actualizarea tehnologică ulterioară.

2.1 Front-End React

React a fost ales pentru partea de front-end datorită modularității, flexibilității și vitezei sale de dezvoltare. Fiind o bibliotecă bazată pe componente, React permite o gestionare eficientă a stării aplicației și facilitarea interacțiunii fluide cu utilizatorul.

În contextul aplicației noastre, structura bazată pe componente oferă un cod ușor de întreținut și extins. Componenta principală, `Home.jsx`, acționează ca un orchestrator, gestionând starea hărții, a căutărilor și a notificărilor.

Aspecte cheie ale implementării Frontend:

- **Virtual DOM:** React utilizează un DOM virtual pentru a optimiza randarea. Atunci când starea aplicației se schimbă (de exemplu, apar noi markere pe hartă), React actualizează doar elementele

modificate din DOM-ul real, asigurând o performanță ridicată chiar și la manipularea a sute de puncte de interes.

- **Integrarea Hărților (React-Leaflet):** Aplicația utilizează biblioteca *Leaflet* prin intermediul wrapper-ului *react-leaflet*. Aceasta permite randarea hărților interactive ca și componente React standard (`<MapContainer>`, `<TileLayer>`, `<Marker>`). Gestionarea evenimentelor de hartă (click, zoom) se face declarativ, integrându-se natural cu ciclul de viață al componentelor React.
- **Managementul Stării (Hooks):** Utilizarea hook-urilor precum `useState` și `useEffect` permite gestionarea logicii complexe (ex: conexiunea WebSocket sau geocodarea asincronă) într-un mod concis și lizibil.
- **Single Page Application (SPA):** Navigarea și interacțiunea se realizează fără reîncărcarea completă a paginii, oferind o experiență similară aplicațiilor native desktop sau mobile.

2.2 Back-End Spring Boot

Spring Boot reprezintă coloana vertebrală a back-end-ului, furnizând un cadru robust și ușor de utilizat pentru dezvoltarea serviciilor. Arhitectura backend-ului este stratificată, respectând principiile *Separation of Concerns*.

Componentele arhitecturale ale backend-ului:

- **Controller Layer:** Gestionează cererile HTTP REST și WebSocket. Acesta acționează ca punct de intrare în aplicație, validând datele și delegând procesarea către serviciile specializate.
- **Service Layer:** Conține logica de business a aplicației. Aici se realizează procesarea datelor despre orașe și recomandări înainte de a fi trimise către client sau salvate în baza de date.
- **Security Layer:** Implementat cu **Spring Security**, acest strat gestionează autentificarea și autorizarea. Configurarea `SecurityFilterChain` asigură protecția endpoint-urilor și gestionează politicile CORS (Cross-Origin Resource Sharing) necesare comunicării cu frontend-ul React.
- **Data Access Layer:** Utilizând **Spring Data JPA**, interacțiunea cu baza de date devine transparentă și eficientă. Spring Boot oferă facilități pentru gestionarea dependențelor (Dependency Injection) și configurarea automată, reducând semnificativ codul "boilerplate".

2.3 Comunicare între Front-End și Back-End

Comunicarea între cele două straturi ale aplicației este vitală și se realizează prin două canale distincte, fiecare servind unui scop specific:

1. API REST (HTTP): Comunicarea standard este gestionată prin cereri HTTP asincrone (folosind `fetch` sau biblioteci precum Axios). Aceasta permite realizarea de operațiuni CRUD eficiente și gestionarea răspunsurilor JSON. Abordarea asincronă contribuie la optimizarea timpului de încărcare; de exemplu, harta este afișată imediat, în timp ce detaliile despre atracții sunt încărcate în fundal.

2. WebSockets (Real-time): Pentru funcționalitățile care necesită actualizare instantanee, cum ar fi notificările, aplicația utilizează protocolul WebSocket.

- **Protocol:** Se utilizează STOMP (Simple Text Oriented Messaging Protocol) peste WebSocket.
- **Implementare:** Clientul React folosește bibliotecile `sockjs-client` și `stompjs` pentru a se abona la topicuri (ex: `/topic/alerts`). Serverul Spring Boot poate trimite mesaje ("push") către acest topic, care ajung instantaneu la toți clienții conectați, fără ca aceștia să efectueze cereri repetate (polling).

2.4 Baza de Date MySQL

Deși inițial s-au luat în considerare diverse soluții, **MySQL** a fost ales ca sistem de gestionare a bazelor de date relaționale datorită robusteții sale, scalabilității și integrării excelente cu ecosistemul Java.

Interfața **Spring Data JPA** (Java Persistence API) simplifică interacțiunea cu baza de date, oferind o abordare elegantă pentru manipularea datelor (ORM - Object Relational Mapping).

- **Entități:** Tabelele din baza de date (ex: `cities`, `recommendations`) sunt mapate direct pe clase Java (`City.java`).
- **Relații:** Relațiile dintre tabele (One-to-Many între Oraș și Recomandări) sunt gestionate automat de Hibernate, asigurând integritatea referențială a datelor.
- **Eficiență:** Interogările SQL sunt generate dinamic, optimizând accesul la date și securizând aplicația împotriva atacurilor de tip SQL Injection.

2.5 Beneficii ale acestei arhitecturi

Alegerea acestui stack tehnologic (React + Spring Boot + MySQL) aduce numeroase avantaje strategice proiectului:

- **Scalabilitate:** Arhitectura decuplată permite extinderea facilă a capacităților aplicației. Backend-ul poate fi scalat orizontal (mai multe instanțe) independent de frontend, pe măsură ce numărul de utilizatori crește.
- **Eficiență în dezvoltare:** Utilizarea framework-urilor consacrate contribuie la o dezvoltare rapidă. React oferă o bibliotecă vastă de componente (hărți, UI), iar Spring Boot automatizează configurările complexe, permițând concentrarea pe logica de business.
- **Performanță optimizată:** Comunicarea eficientă între front-end și back-end, împreună cu utilizarea unei baze de date robuste și a mecanismelor de caching ale browserului, asigură o performanță optimă. Hărțile se încarcă rapid datorită randării eficiente din React.
- **Flexibilitate și extensibilitate:** Modularitatea și structura componentelor permit o adaptabilitate ușoară. Adăugarea unei noi funcționalități, cum ar fi un sistem de recenzii, ar implica crearea unei noi componente React și a unui nou endpoint în Controller, fără a afecta restul aplicației.

Arhitectura noastră reflectă angajamentul față de o experiență utilizator de calitate și o dezvoltare durabilă. Prin integrarea tehnologiilor de vârf precum WebSockets și hărți interactive, am construit o aplicație care satisface nevoile actuale de informare turistică rapidă și vizuală.

3 Implementare Frontend

În dezvoltarea front-end-ului aplicației **AI Travel Guide**, am adoptat o abordare meticuloasă pentru a oferi utilizatorilor o experiență vizuală imersivă și intuitivă. Front-end-ul reprezintă interfața principală prin care utilizatorii interacționează cu platforma noastră, iar alegerea tehnologiilor potrivite și implementarea unei structuri coerente au fost priorități cheie.

Provocarea principală a constat în sincronizarea datelor geografice complexe (coordonate, hărți, markere) cu informațiile descriptive (text, imagini) și actualizările în timp real, menținând în același timp o performanță ridicată a browserului și o fluiditate a navigării.

3.1 Tehnologii frontend

Am optat pentru **React** ca bibliotecă principală pentru dezvoltarea front-end-ului datorită modularității sale și ecosistemului vast. Integrarea React ne-a permis să construim o interfață de utilizator dinamică de tip Single Page Application (SPA), unde reîncărcarea paginii este eliminată, oferind o experiență similară aplicațiilor native.

Tehnologiile specifice și motivarea alegerii lor includ:

- **React (Hooks Functional Components):** Am utilizat exclusiv componente funcționale și Hooks (`useState`, `useEffect`, `useRef`) pentru a gestiona starea aplicației. Această abordare modernă simplifică logica și permite reutilizarea codului. De exemplu, `useRef` a fost esențial pentru a menține referințe către instanța hărții Leaflet fără a declanșa randări inutile.
- **React-Leaflet:** Această bibliotecă acționează ca o punte între React și Leaflet (motorul de hărți). Ea ne permite să declarăm hărțile și markerele ca și componente React standard, gestionând automat ciclul de viață al hărții (inițializare, actualizare, distrugere) în funcție de schimbările de stare din aplicație.
- **Axios:** Pentru comunicarea cu serverul, am ales Axios datorită capacității sale de a gestiona cereri HTTP bazate pe Promisiuni. Acesta facilitează interceptarea cererilor, gestionarea erorilor și transformarea automată a datelor JSON, fiind folosit intens în serviciile noastre de date.
- **SockJS și StompJS:** Pentru a implementa funcționalitatea de notificări în timp real, am integrat aceste biblioteci care permit stabilirea unei conexiuni WebSocket stabile și abonarea la topicuri de mesaje, asigurând compatibilitatea chiar și cu browserele mai vechi.

3.2 Structura componentelor

Structura proiectului este organizată modular, respectând principiul separării responsabilităților (Separation of Concerns). Logica de prezentare (UI) este strict separată de logica de comunicare cu serverul (Servicii) și de logica de stilizare (CSS).

3.2.1 Componentele și Serviciile din Proiect

Principalele fișiere și componente implementate sunt centralizate în tabelul următor, reflectând structura logică a aplicației:

Nume Fișier / Componentă	Rolul componentei	Ruta / Utilizare
`Home.jsx`	Componenta principală (Dashboard). Gestionează harta, căutarea, logica Wikipedia și WebSockets.	`/`
`Login.jsx`	Formularul de autentificare a utilizatorilor.	`/login`
`Register.jsx`	Formularul de înregistrare a noilor utilizatori.	`/register`
`LiveRecommendations.jsx`	Componentă dedicată afișării notificărilor în timp real (Toast notifications).	Importată în Home
`ChangeView`	Componentă internă (în Home.jsx) pentru recentrarea hărții la schimbarea coordonatelor.	Helper Map
`LocationEvents`	Componentă internă (în Home.jsx) pentru gestionarea click-urilor pe hartă.	Helper Map
`recommendationService.jsx`	Serviciu care abstractizează apelurile API pentru orașe și recomandări.	Service
`authService.jsx`	Serviciu pentru apeluri API de autentificare (Login/Signup).	Service

Tabela 2: Structura componentelor Frontend

3.3 Descrierea Detaliată a Componentelor Principale

3.3.1 Home.jsx - Orchestratorul Aplicației

Componenta `Home.jsx` este inima aplicației și cea mai complexă piesă din arhitectura frontend. Aceasta nu este o simplă pagină de afișare, ci un orchestrator care gestionează multiple fluxuri de date simultan.

Gestionarea Hărții și a Stării Vizuale: Una dintre responsabilitățile principale ale componentei este sincronizarea hărții cu acțiunile utilizatorului. Deoarece biblioteca Leaflet funcționează imperativ (prin comenzi directe către hartă), iar React funcționează declarativ (prin schimbarea stării), am implementat o componentă intermediară numită `ChangeView`. Aceasta ascultă modificările coordonatelor din starea aplicației și comandă hărții să se animeze ("flyTo") către noua locație, asigurând o tranziție fluidă între orașe.

Logica de Căutare și Îmbogățire a Datelor (Wikipedia): Procesul de căutare este unul sofisticat. Când utilizatorul caută un oraș, componenta execută următorii pași secvențiali:

1. Interoghează backend-ul propriu pentru a obține lista de atracții turistice stocate.
2. Utilizează un serviciu de geocodare pentru a obține coordonatele exacte ale orașului și centrează harta.
3. Pentru fiecare atracție primită, lansează o căutare asincronă către API-ul Wikipedia. Aici intervine o logică complexă de filtrare: se verifică dacă titlul articolului Wikipedia corespunde cu numele atracției și dacă coordonatele geografice ale articolului sunt în apropierea orașului căutat. Aceasta previne erorile de afișare (de exemplu, afișarea unui muzeu cu același nume din altă țară).

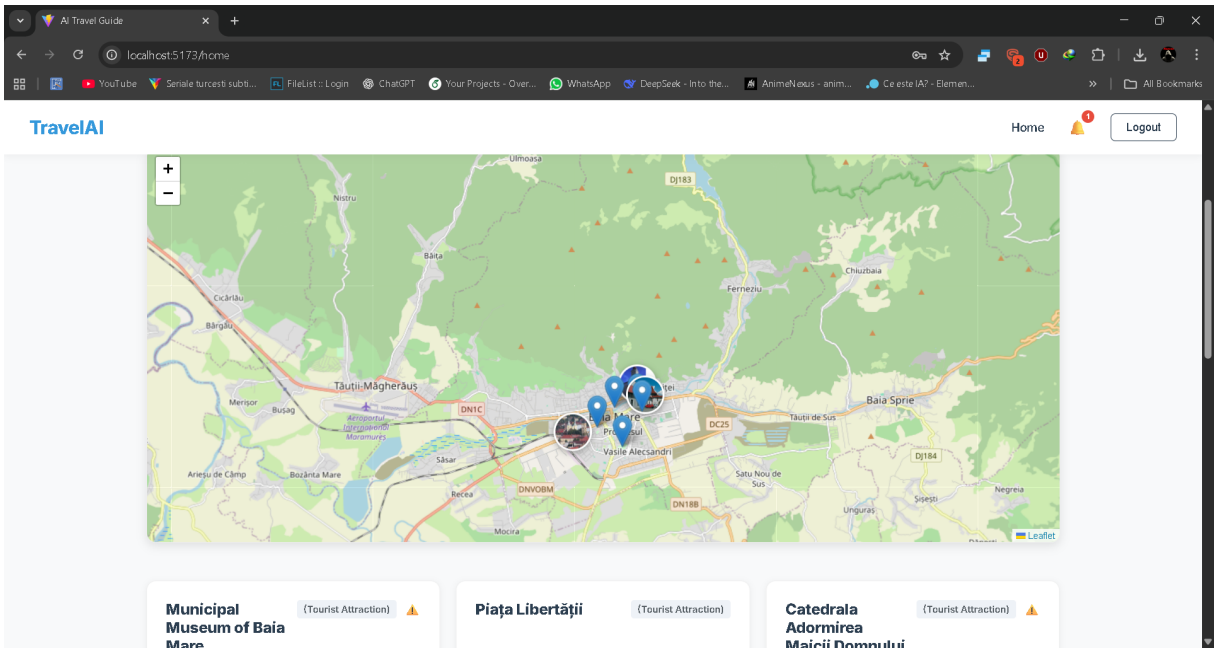

Figura 1: Harta interactivă populată cu markere

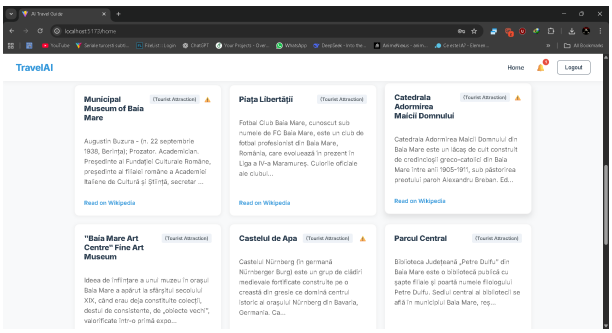

Figura 2: Caption

3.3.2 LiveRecommendations.jsx - Sistemul de Notificări

Această componentă este responsabilă pentru feedback-ul vizual în timp real. Ea funcționează independent de restul interfeței, fiind suprapusă (overlay) peste conținutul principal.

Componenta menține o listă internă de notificări active. Când primește un mesaj nou prin WebSocket (transmis de componenta părinte `Home`), aceasta creează un element vizual de tip "Toast" care apare discret în colțul ecranului. Fiecare notificare are un temporizator intern care o elimină automat după câteva secunde, asigurând că ecranul utilizatorului nu devine aglomerat. De asemenea, utilizatorul poate închide manual notificările.

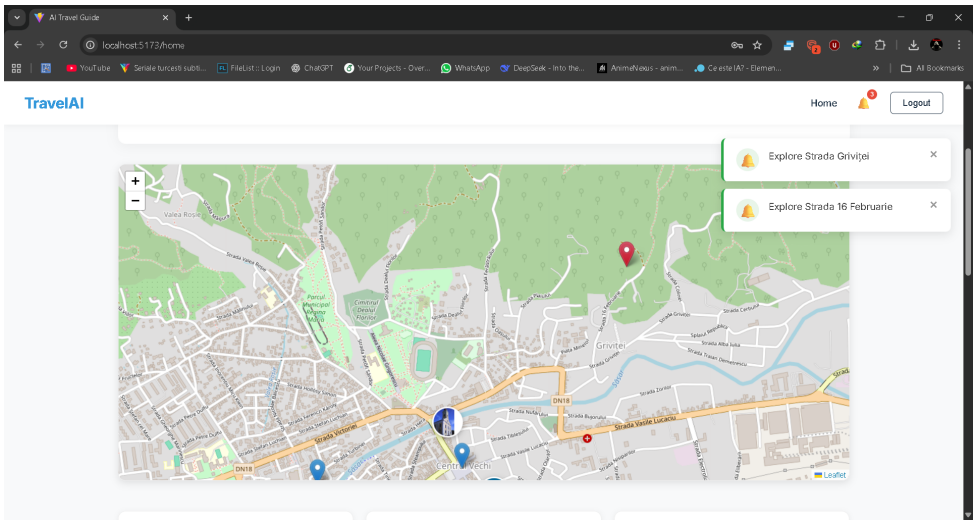

Figura 3: Afișarea notificărilor în timp real

3.3.3 Serviciile de Date (recommendationService și authService)

Pentru a respecta bunele practici de inginerie software, nicio componentă vizuală nu face apeluri directe către URL-uri hardcodate. Toate interacțiunile cu rețeaua sunt încapsulate în module de servicii.

`recommendationService.jsx` acționează ca un strat de abstracție peste biblioteca Axios. Acesta definește metode clare pentru obținerea orașelor, a recomandărilor specifice și a sugestiilor bazate pe locație. Acest design permite modificarea ușoară a endpoint-urilor API sau adăugarea de logică de caching fără a afecta componentele vizuale care consumă aceste date.

Similar, `authService.jsx` gestionează ciclul de viață al autentificării, ocupându-se de trimiterea credențialelor și, potențial, de stocarea token-urilor de sesiune în LocalStorage.

3.3.4 Login.jsx și Register.jsx - Gestiunea Accesului

Aceste componente oferă poarta de intrare pentru utilizatorii care doresc o experiență personalizată. Designul lor este centrat pe simplitate și validare.

Formularele implementate utilizează starea locală pentru a monitoriza input-urile utilizatorului în timp real (Controlled Components). În momentul trimiterii formularului, datele sunt validate sumar pe client înainte de a fi trimise către serviciul de autentificare. Componentele gestionează, de asemenea, stările de eroare, afișând mesaje clare utilizatorului în cazul în care credențialele sunt incorecte sau dacă serverul nu răspunde.

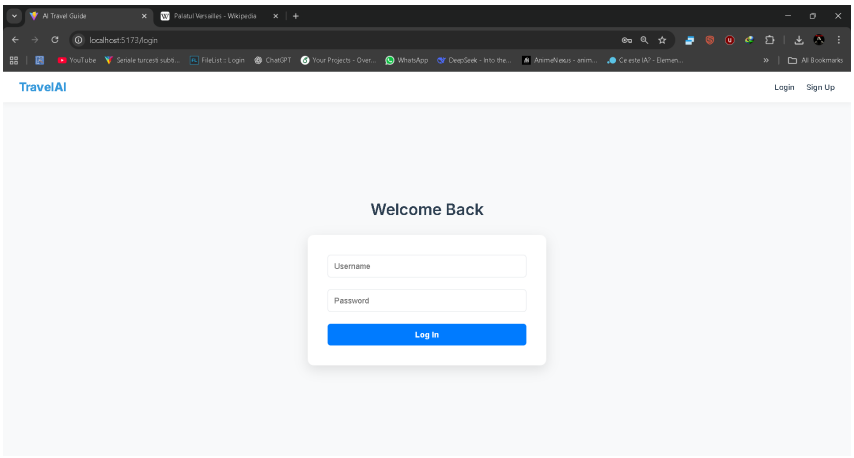

Figura 4: Interfața de autentificare a utilizatorului

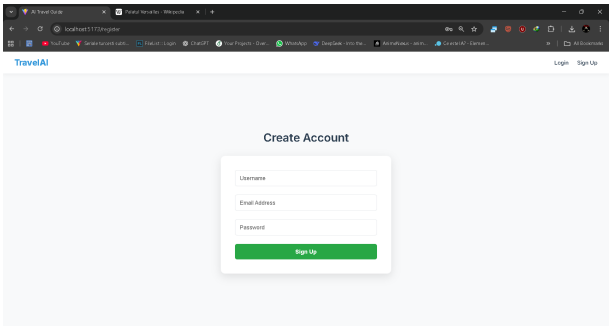

Figura 5: Pagina de inregistrare

3.4 Fluxul de Date și Interactivitatea

Experiența utilizatorului este definită de fluiditatea cu care datele circulă între aceste componente.

Fluxul de Căutare: Când utilizatorul inițiază o căutare în `Home.jsx`, starea `loading` este activată, afișând un indicator vizual. Datele sunt preluate asincron, iar pe măsură ce sosesc (mai întâi coordonatele orașului, apoi lista de atracții, apoi imaginile de pe Wikipedia), interfața se actualizează progresiv. Harta se centrează automat, iar markerele apar unul câte unul, oferind un sentiment de dinamism.

Fluxul de Notificări: Conexiunea WebSocket este stabilită o singură dată, la montarea componentei `Home`. Această conexiune persistentă ascultă canalul `/topic/alerts`. Când un mesaj sosește, acesta este propagat imediat către componenta `LiveRecommendations` și către contorul din bara de navigare, asigurând că utilizatorul nu pierde nicio informație critică, indiferent de acțiunea pe care o desfășoară în acel moment.

4 Implementare Backend

Backend-ul aplicației **AI Travel Guide** reprezintă nucleul logic și funcțional al sistemului. Acesta este responsabil pentru gestionarea securității, persistența datelor, orchestrarea logicii de business și facilitarea comunicării în timp real. Arhitectura backend-ului a fost proiectată urmând principiile *Layered Architecture* (Arhitectură Stratificată), asigurând o separare clară a responsabilităților și o mentenabilitate ridicată.

Implementarea a fost realizată folosind limbajul **Java** și ecosistemul **Spring Boot**, o alegere motivată de robustețea, scalabilitatea și suportul vast al comunității pentru aceste tehnologii în mediul enterprise.

4.1 Arhitectura și Tehnologiile Utilizate

4.1.1 Spring Boot și Inversion of Control (IoC)

Spring Boot a fost ales ca framework principal datorită capacității sale de a simplifica configurarea aplicațiilor bazate pe Spring. Un concept fundamental utilizat în întreaga aplicație este **Inversion of Control (IoC)**, implementat prin mecanismul de **Dependency Injection (DI)**.

În loc ca obiectele să își creeze singure dependențele (de exemplu, un Controller să instanțieze manual un Service), acestea declară dependențele necesare prin constructor, iar containerul Spring (ApplicationContext) se ocupă de injectarea instanțelor la runtime.

Avantajele acestei abordări în proiect:

- **Decuplare:** Componentele sunt slab cuplate (loose coupling), ceea ce face codul mai modular.
- **Testabilitate:** Dependențele pot fi ușor înlocuite cu obiecte "mock" în timpul testării unitare.
- **Singleton Pattern:** Spring gestionează implicit fasolele (beans) ca singleton-uri, optimizând utilizarea memoriei.

4.1.2 Maven - Gestionarea Dependențelor

Pentru managementul proiectului și al bibliotecilor, am utilizat **Apache Maven**. Fișierul `pom.xml` definește structura proiectului și dependențele externe. Printre cele mai importante dependențe integrate se numără:

- `spring-boot-starter-web`: Pentru crearea aplicațiilor web RESTful, incluzând serverul embedded Tomcat.
- `spring-boot-starter-data-jpa`: Pentru persistența datelor folosind Hibernate.
- `spring-boot-starter-security`: Pentru autentificare și autorizare.
- `spring-boot-starter-websocket`: Pentru comunicarea în timp real.
- `mysql-connector-j`: Driverul JDBC pentru conexiunea la baza de date MySQL.
- `lombok`: O bibliotecă care reduce codul "boilerplate" (getters, setters, constructori) prin adnotări.

4.2 Modelarea Datelor și Persistența (JPA)

Stratul de persistență este construit pe baza standardului **JPA (Java Persistence API)**, utilizând **Hibernate** ca implementare ORM (Object-Relational Mapping). Această tehnologie ne permite să manipulăm datele din baza de date MySQL folosind obiecte Java (POJO), eliminând necesitatea scrierii manuale a interogărilor SQL complexe.

4.2.1 Entitatea City

Entitatea `City` reprezintă punctul central al modelului de date geografic. Aceasta este mapată la tabela `cities` din baza de date.

Listing 1: Definirea Entității City

```
@Entity
@Table(name = "cities")
public class City {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String name;

    @OneToMany(mappedBy = "city", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Recommendation> recommendations = new ArrayList<>();

    // Constructori, Getters i Setters genera i sau defini i manual
}
```

Analiza tehnică a implementării:

- `@Entity`: Marchează clasa ca fiind o entitate JPA gestionată de Hibernate.
- `@Table`: Specifică numele tabelei din baza de date.
- `@Id` și `@GeneratedValue`: Definesc cheia primară și strategia de auto-incrementare a acesteia (IDENTITY), delegând generarea ID-ului către MySQL.
- `@OneToMany`: Această adnotare definește relația fundamentală a aplicației. Un oraș poate avea mai multe recomandări turistice.
  - `mappedBy = "city"`: Indică faptul că relația este bidirecțională și că entitatea `Recommendation` este cea care deține cheia străină (owner side).
  - `cascade = CascadeType.ALL`: Asigură propagarea operațiunilor. Dacă un oraș este salvat, sunt salvate și recomandările lui. Dacă este șters, sunt șterse și recomandările.
  - `orphanRemoval = true`: Garantează integritatea datelor; dacă o recomandare este eliminată din lista orașului, ea va fi ștearsă fizic din baza de date.

4.2.2 Entitatea Recommendation

Această entitate stochează detaliile specifice ale atracțiilor turistice, restaurantelor sau evenimentelor asociate unui oraș.

Listing 2: Structura Entității Recommendation

```
@Entity
@Table(name = "recommendations")
public class Recommendation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(length = 1000) // Extindem limita standard de 255 caractere
    private String description;
```

```
    private String category;
    private String imageUrl;
    private Double latitude;
    private Double longitude;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "city_id", nullable = false)
    private City city;
}
```

Utilizarea `@ManyToOne` cu `FetchType.LAZY` (implicit sau explicit) este o optimizare de performanță. Aceasta înseamnă că datele despre oraș nu sunt încărcate din baza de date decât atunci când sunt explicit accesate, reducând încărcarea memoriei atunci când interogăm doar lista de recomandări.

4.2.3 Repository Pattern

Accesul la date este abstractizat prin interfețe care extind `JpaRepository`. Spring Data JPA generează automat implementarea acestor interfețe la runtime, oferind metode standard CRUD (save, findAll, findById, delete) și permițând definirea de interogări derivate din numele metodelor.

```
public interface CityRepository extends JpaRepository<City, Long> {
    Optional<City> findByName(String name);
}
```

Metoda `findByName` este tradusă automat de Spring într-un query SQL de tipul: `SELECT * FROM cities WHERE name = ?`.

4.3 Logica de Business și Servicii

Stratul de servicii (`Service Layer`) conține logica propriu-zisă a aplicației. Acesta acționează ca un intermediar între Controller și Repository, asigurând validarea datelor și procesarea tranzacțională.

4.3.1 RecommendationService

Acest serviciu gestionează logica legată de destinațiile turistice. Deși în codul sursă furnizat logica este simplificată, arhitectura permite extinderea ușoară pentru scenarii complexe, cum ar fi filtrarea recomandărilor în funcție de preferințele utilizatorului sau calcularea distanțelor geografice.

4.3.2 AuthService și UserDetailsServiceImpl

Pentru autentificare, am implementat interfața standard `UserDetailsService` din Spring Security. Clasa `UserDetailsServiceImpl` este responsabilă pentru încărcarea datelor utilizatorului din baza de date pe baza numelui de utilizator (username).

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundExcep
        User user = userRepository.findByUsername(username)
```

```
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));
        return new org.springframework.security.core.userdetails.User(
            user.getUsername(),
            user.getPassword(),
            new ArrayList<>() // Roluri/Autorit i
        );
    }
}
```

Această implementare permite integrarea perfectă a modelului nostru de date (`User`) cu mecanismele interne de securitate ale Spring.

4.4 Securitate și Controlul Accesului

Securitatea aplicației **AI Travel Guide** este un aspect critic, gestionat prin framework-ul **Spring Security**. Configurarea este centralizată în clasa `SecurityConfig`, care definește lanțul de filtre de securitate (`SecurityFilterChain`).

4.4.1 Mecanismul de Autentificare

Am utilizat `DaoAuthenticationProvider`, care este provider-ul standard pentru autentificarea bazată pe baze de date. Acesta colaborează cu:

1. **UserDetailsService:** Pentru a găsi utilizatorul.
2. **PasswordEncoder:** Pentru a verifica parola.

Criptarea Parolelor (BCrypt): Stocarea parolelor în clar este o vulnerabilitate majoră. Am configurat un bean de tip `BCryptPasswordEncoder`.

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

BCrypt este un algoritm de hashing adaptiv, care include automat un "salt" (șir aleatoriu) pentru a proteja împotriva atacurilor de tip dicționar sau rainbow tables.

4.4.2 Configurarea CORS (Cross-Origin Resource Sharing)

O provocare specifică arhitecturilor decuplate (Frontend React separat de Backend Spring) este politica CORS a browserelor. Deoarece frontend-ul rulează pe `localhost:5173` și backend-ul pe `localhost:8080`, browserul consideră cererile ca fiind "cross-origin" și le blochează implicit.

Am definit o configurație CORS explicită în `SecurityConfig`:

```
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(Arrays.asList("http://localhost:5173", "http://local
    configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIO
    configuration.setAllowCredentials(true);
    // ...
}
```

Aceasta permite frontend-ului să trimită cereri HTTP și să includă credențiale (cookie-uri de sesiune) în aceste cereri.

4.4.3 Autorizarea Endpoint-urilor

Am adoptat o strategie de securitate de tip "whitelist", unde totul este interzis implicit, cu excepția rutelor specificate.

- `/api/v1/auth/**`: Permis tuturor (Login, Register).
- `/api/v1/recommendation/**`: Permis tuturor (pentru a permite vizitatorilor să vadă harta).
- `/ws/**`: Permis tuturor (pentru handshake-ul WebSocket).
- `anyRequest().authenticated()`: Orice altceva necesită autentificare.

4.5 API REST și Controlere

Stratul de prezentare al backend-ului este expus prin API-uri RESTful. Controlerele sunt adnotate cu `@RestController`, ceea ce înseamnă că răspunsurile sunt serializate automat în format JSON.

4.5.1 RecommendationController

Acest controller gestionează cererile legate de datele turistice. Un aspect interesant al implementării este endpoint-ul `/geocode`.

```
@GetMapping("/geocode")
public Map<String, Double> geocodeCity(@RequestParam String city) {
    // Logic de proxy c tre un serviciu extern sau baza de date
}
```

Acest endpoint acționează ca un **Proxy**. Deși frontend-ul ar putea apela direct API-uri externe de geocodare, trecerea cererii prin backend oferă avantaje:

- **Securitate:** Putem ascunde cheile API ale serviciilor externe pe server.
- **CORS:** Evităm problemele de CORS pe care le au unele API-uri externe.
- **Caching:** Putem salva rezultatele geocodării în baza de date pentru a nu apela serviciul extern de fiecare dată.

4.5.2 DTO (Data Transfer Objects)

Pentru transferul datelor între client și server, am utilizat pattern-ul DTO. De exemplu, pentru login, folosim clasa `LoginRequest`:

```
public class LoginRequest {
    private String userName;
    private String password;
    // getters & setters
}
```

Utilizarea DTO-urilor, în loc de a expune direct entitățile JPA, decuplează structura internă a bazei de date de contractul API-ului extern, permițând modificarea bazei de date fără a "strica" frontend-ul.

4.6 Comunicarea în Timp Real (WebSockets)

Un element distinctiv al aplicației este utilizarea WebSockets pentru notificări. Spre deosebire de HTTP, care este un protocol "Request-Response" (clientul cere, serverul răspunde), WebSocket permite o conexiune persistentă, full-duplex.

4.6.1 Protocolul STOMP

Deoarece WebSocket este un protocol de nivel scăzut, am utilizat **STOMP (Simple Text Oriented Messaging Protocol)** peste WebSocket. Acesta definește un format standard pentru mesaje (SUBSCRIBE, SEND), facilitând rutarea mesajelor.

Fluxul de Notificare:

1. **Handshake:** Clientul React inițiază conexiunea la `http://localhost:8080/ws`.
2. **Subscription:** Clientul se abonează la topicul `/topic/alerts`.
3. **Broadcast:** Când serverul dorește să trimită o alertă (ex: vreme rea), trimite un mesaj către acest topic. Broker-ul de mesaje intern al Spring distribuie mesajul către toți clienții abonați activi.

Această arhitectură este extrem de scalabilă și permite implementarea unor funcționalități avansate precum chat live, actualizări de locație în timp real sau alerte de urgență.

4.7 Configurarea Mediului și Baza de Date

Configurarea aplicației este centralizată în fișierul `application.properties`.

Conexiunea la Baza de Date:

```
spring.datasource.url = jdbc:mysql://localhost:3306/proiect_paw?createDatabaseIfNotExist
spring.datasource.username = root
spring.datasource.password = password
```

Parametrul `createDatabaseIfNotExist=true` asigură că baza de date este creată automat la prima rulare, simplificând procesul de instalare.

Hibernate DDL Auto:

```
spring.jpa.hibernate.ddl-auto=update
```

Setarea `update` instruiește Hibernate să verifice schema bazei de date la pornire și să o actualizeze dacă entitățile Java s-au modificat (de exemplu, s-a adăugat o coloană nouă), fără a șterge datele existente. Aceasta este ideală pentru mediul de dezvoltare.

Gestionarea Sesiunilor:

```
server.servlet.session.cookie.same-site=None
server.servlet.session.cookie.secure=false
```

Aceste setări sunt necesare pentru a permite cookie-urilor de sesiune să funcționeze într-un mediu cross-site (Frontend și Backend pe porturi diferite) fără HTTPS, specific fazei de dezvoltare locală.

În concluzie, implementarea backend-ului demonstrează o utilizare matură a ecosistemului Java Enterprise. Arhitectura aleasă asigură securitatea datelor, performanța prin optimizări JPA și o experiență utilizator superioară prin capabilitățile de timp real.

5 Concluzii

Aplicația **AI Travel Guide** reprezintă o soluție modernă și scalabilă pentru industria turismului digital, demonstrând cum integrarea datelor geografice cu informații contextuale poate transforma experiența de planificare a unei călătorii. Proiectul a reușit să atingă obiectivele propuse, oferind o interfață vizuală intuitivă și un backend robust capabil să gestioneze date complexe.

Posibilitățile de dezvoltare în cadrul acestei aplicații sunt ample și promițătoare, oferind oportunități pentru îmbunătățiri continue și extinderea funcționalităților într-un ecosistem digital în continuă schimbare.

5.1 Realizări și Impact

Proiectul a demonstrat viabilitatea unei arhitecturi decuplate (Frontend-Backend) în contextul aplicațiilor geospațiale. Utilizarea bibliotecii **Leaflet** în cadrul ecosistemului **React** a permis crearea unei experiențe de utilizare fluide, unde harta nu este doar o imagine statică, ci un instrument interactiv de explorare.

Pe partea de server, **Spring Boot** s-a dovedit a fi o alegere excelentă, gestionând eficient logica de business și comunicarea cu baza de date **MySQL**. Implementarea mecanismelor de securitate și a protocolului WebSocket a adăugat un strat de profesionalism și dinamism aplicației.

5.2 Direcții de Dezvoltare Viitoare

Începând cu cele mai mici îmbunătățiri ale aplicației, există un potențial vast de creștere:

- **Personalizare Avansată:** Implementarea posibilității ca utilizatorii să își creeze conturi, să salveze liste de "Favorite" și să construiască itinerarii personalizate pe zile. De exemplu, un utilizator ar putea selecta 5 muzee și aplicația să genereze traseul optim între ele.
- **Conținut Multimedia Extins:** Extinderea integrării cu API-uri externe pentru a include nu doar imagini de pe Wikipedia, ci și galerii video scurte (ex: YouTube API) sau tururi virtuale 360 de grade pentru obiectivele majore.
- **Sistem de Recenzii și Comunitate:** Adăugarea unei componente sociale unde turiștii pot lăsa recenzii, pot încărca propriile fotografii și pot acorda note atracțiilor vizitate, creând astfel o bază de date organică, întreținută de comunitate.
- **Integrare AI Generativă:** Utilizarea unor modele de limbaj (LLM) precum OpenAI pentru a genera descrieri turistice personalizate în funcție de interesele utilizatorului (ex: "Arată-mi locuri romantice în Paris") sau pentru a funcționa ca un ghid audio virtual.

Prin integrarea tehnologiei **WebSocket** pentru notificări în timp real, s-a deschis o poartă către un mediu mult mai dinamic. În prezent, sistemul poate alerta utilizatorii despre evenimente generale, dar există perspective clare pentru:

- **Alerte Contextuale:** Notificări bazate pe geolocația exactă a utilizatorului (ex: "Ai trecut pe lângă o cafenea istorică").
- **Gamification:** Introducerea unor elemente de joc, cum ar fi "vânătoarea de comori" urbană, unde utilizatorii deblochează badge-uri vizitând anumite locații de pe hartă, stimulând astfel angajamentul și explorarea fizică.
- **Analiză Predictivă:** Utilizarea datelor istorice pentru a recomanda cel mai bun moment al zilei pentru vizitarea unui obiectiv, evitând aglomerația.

5.3 Sinteză Finală

În concluzie, întreaga aplicație reprezintă un ansamblu coerent și eficient de tehnologii și funcționalități menite să ofere o experiență avansată și personalizată pentru turiști. Implementarea front-end-ului cu **React**, backend-ul cu **Spring Boot** și integrarea **WebSockets** aduc un nivel înalt de interactivitate, eficiență și comunicare în timp real.

Capacitatea de a agrega date din surse externe (Wikipedia, OpenStreetMap) și de a le prezenta într-un format unificat demonstrează puterea interoperabilității în web-ul modern. Utilizarea **Spring Security** pentru protecția datelor, a bazei de date **MySQL** pentru persistență și a protocolului STOMP pentru comunicarea în timp real subliniază maturitatea tehnică a soluției.

AI Travel Guide nu este doar un instrument de informare, ci o platformă extensibilă care poate evolua odată cu nevoile utilizatorilor și cu noile tehnologii din domeniul Travel Tech, facilitând procesul de descoperire a lumii într-un mod accesibil și interactiv.

6 Bibliografie

1. **Spring Boot Documentation**, 2024. Disponibil online:
   `https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/`

2. **React Documentation**, 2024. Disponibil online:
   `https://react.dev/`

3. **Leaflet JS Documentation**. Disponibil online:
   `https://leafletjs.com/reference.html`

4. **React Leaflet Documentation**. Disponibil online:
   `https://react-leaflet.js.org/`

5. **MySQL Reference Manual**. Disponibil online:
   `https://dev.mysql.com/doc/refman/8.0/en/`

6. **OpenStreetMap Nominatim API**. Disponibil online:
   `https://nominatim.org/release-docs/latest/api/Overview/`

7. **MediaWiki Action API (Wikipedia)**. Disponibil online:
   `https://www.mediawiki.org/wiki/API:Main_page`

8. **Spring Security Architecture**. Disponibil online:
   `https://spring.io/guides/topicals/spring-security-architecture`

9. **WebSocket Support in Spring**. Disponibil online:
   `https://docs.spring.io/spring-framework/reference/web/websocket.html`